readme.md

Behavioral Cloning

Behavioral Cloning Project

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around the track without leaving the road

Rubric Points

The project requirements are listed here rubric points

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

I have included the following

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- nvidia_cloning_model.h5 containing a trained convolution neural network for track one
- nvidia_cloning_model_track2.h5 containing a trained convolution neural network for track two
- parameters.json containing training and data augmentation parameters
- DataAugmentation.py implentation of data augmentation techniques for images
- writeup_report.mdsummarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around:

- Track one by executing

```
python drive.py nvidia_cloning_model.h5
```

- Track two using

```
python drive.py nvidia_cloning_model_track2.h5
```

3. Submission code is usable and readable

model.py contains two convolutional neural networks, LeNet and DAVE-2. It includes the entire training pipeline from loading the data, augmentation techniques, training (either network or transfer learning from track one to track two) and validating the network.

The image preprocessing and augmentation is implemented in DataAugmentation.py

Training and data augmentation parameters are defined in parameters.json

This faciliated the training process on the AWS cloud instance as several changes and parameters had to be tuned.

4. Video of the vehicle driving around autonomously

- Track one
- Track two

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

DAVE-2 was implemented (model.py lines 199-216)

The model includes 4 CNN layers with pooling layers in between, followed by four fully connected layers. The original model includes an additional normalisation layer which was implemented using a Keras lambda layer (code line 195), as shown later on.

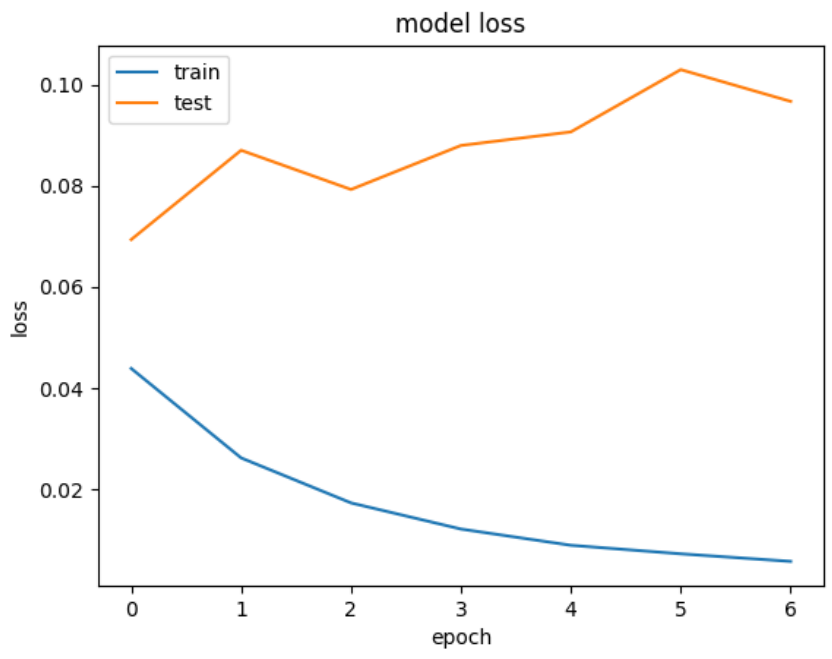
The model includes RELU layers to introduce nonlinearity (code line 208) and a tanh activation function, which introduces nonlinearlity to the model.

I will not go into a discussion on LeNet has it has been implemented before, additionally it was'nt not used to generate the results used here.

2. Attempts to reduce overfitting in the model

The model was trained and validated (20% split) on different data sets to ensure that the model was not overfitting. The data was shuffled before splitting into training and validation, as it was collected chronologically by driving the vehicle in the simulator. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. The training and validation loss was monitored to avoid over training, by inspecting the parameter.json file it can be seen that the epochs were limited to 7 as the validation loss was stagnating.

The training and validation losses are shown below for the seven epochs used for training:



3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (code line 231).

Additionally, using a parameter (epochs, augmentation techinques, multiple datasets, networks...etc) file enabled rapid prototyping of the model on the AWS instance without having to inspect the entire code based which was a few hundred lines of code.

4. Appropriate training data

For the first track, three datasets were created. 1- Counter clockewise loop, which is the required task.

It was clear that the drive was dominated by counter clock wise turns.

2- Clock wise loop: to unify the training data distrubition.

3- Recovery: Created for specialy cases such as driving straight on the bridge, recovery from extreme left and right cases, turning away from dirt roads, etc.

4- Data augemtnation: certain data augmentation techinques were applied to all datasets, which will be discussed in more detail.

Model Architecture and Training Strategy

1. Solution Design Approach

It was clear that there was an existing body for research on this work and that it would be not be pratical to build and design a network from scratch. CNNs were the obvious choice for image processing applications. I implemented both LeNet and DAVE-2. The latter was used to solve this very problem for real world driving.

I did not use LeNet as the training time was a lot longer than DAVE-2 and DAVE-2 seemed to be more suited for the problem at hand.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set.

I initally had good results in terms of MSE using the counter clock wise dataset mentione earlier however, the vehicle was driving well only on clockwise turns. I followed by adding a clock wise dataset. The data augmentation techinques will be discussed later on.

I found the MSE was not a good indication of the driving performance, I could only validate the performance of the model by driving the vehicle.

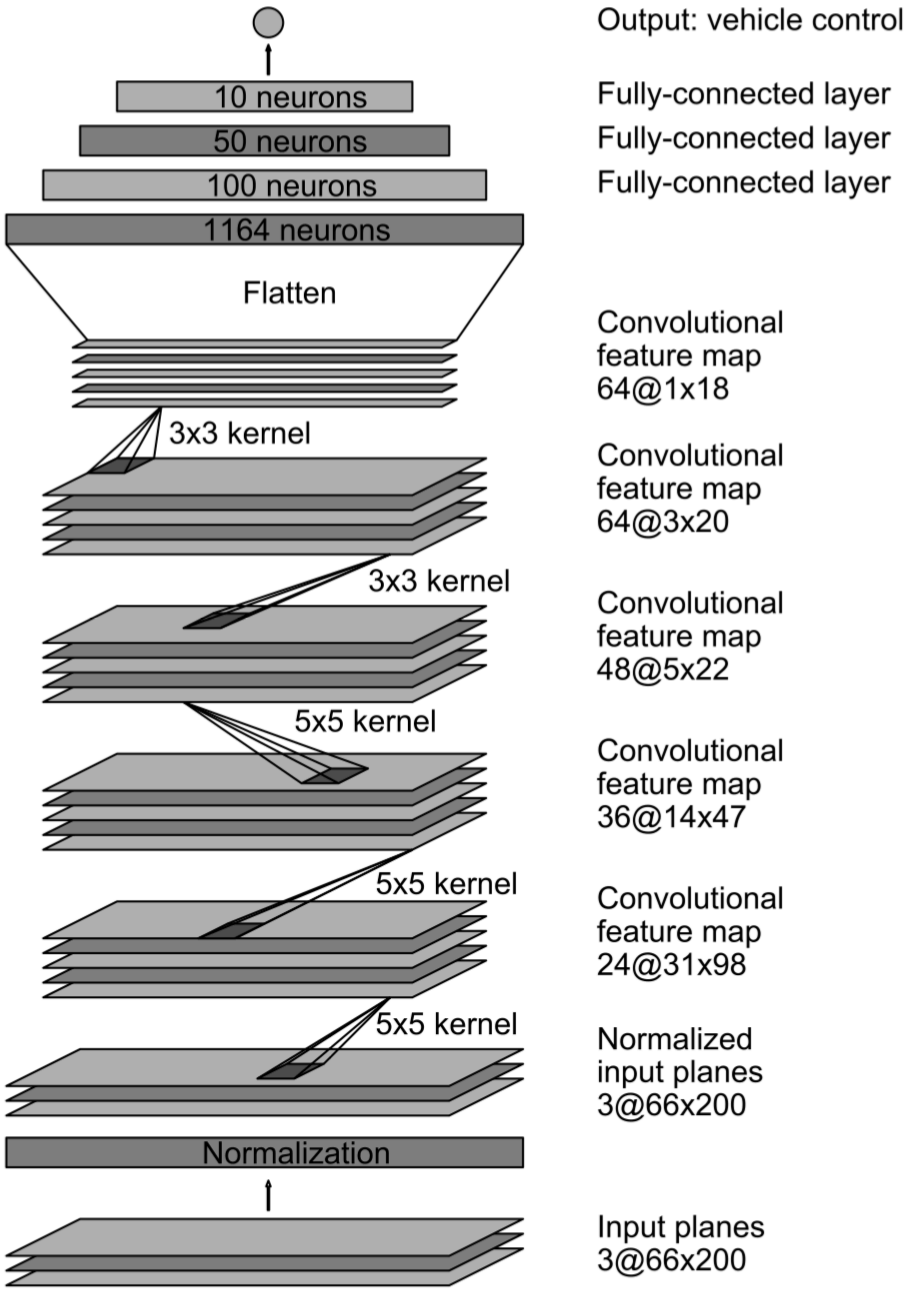
At the end of the process, the vehicle is able to drive autonomously around both track without leaving the road

2. Final Model Architecture

The final model architecture model architecture was DAVE-2

```
model.add(Conv2D(24,5,5, activation='relu', subsample=(2,2)))
model.add(Conv2D(36,5,5, activation='relu', subsample=(2,2)))
model.add(Conv2D(48,5,5, activation='relu', subsample=(2,2)))
model.add(Conv2D(64,3,3, activation='relu', subsample=(1,1)))
model.add(Conv2D(64,3,3, activation='relu', subsample=(1,1)))
model.add(Flatten())
model.add(Dense(1164))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('tanh'))
```

Here is a visualization of the architecture from the original paper



3. Final Data Augmentation, Training Datasets and Parameters

The final parameters used were as follows:

```
{
 "data_sets": ["data/counterclockwise/", "data/clockwise/", "data/recovery/"],
 "log_name": "driving_log.csv",
 "data_augmentation":
               {
                   "flip":true,
                   "use_left_camera": true,
                   "use_right_camera": true,
                   "camera_correction": 0.3,
                   "flip_images": true,
                   "darken": false,
                   "dark_gamme":0.35,
                   "brighten": false,
                   "bright_gamma":5,
                   "blur":false,
                   "blur_kernel":5,
                   "rotate":false,
                   "rotation_angle_deg": 5,
                    "translate":false,
                    "distance_px":10,
                    "randomise": false,
                    "randomise_factor":1,
                    "show_images": false,
                    "write_images": false

                },

  "training":
                {
                    "epochs": 7,
                    "network": "nvidia",
                    "visualise_performance": true,
                    "load_model": false,
                    "model_name": "nvidia_cloning_model.h5"
                }
}
```

Two parameter to note here, is that I did not apply rotation and translation to the images as recommended by the DAVE-2 creaters as that requried the a correction to the steering angle which was another hyper parameter to tune, without the correction factor the performance seems to degrade.

Additionally, I did not convert the images to YUV space as recommended by the authors. I did not seem to gain any improvements.

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I recorded a single clockwise loop for track one and a recovery lap.

For track two, I only captured a single lap which was not used in the inital training run.

To augment the data sat I attempted darkning, brighting and blurring the image but that did not improve the performance. My thought was this would train the model to identify that curvature of the road is the only factor that influences the steering angle.

The images were flipped and corresponding steering angles were negated.

The data was grayscaled and normalised.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

5. Transfer Learning to track two

For track two, I implemented a fine tuning strategy as the trained model for track one would be suitable. I only used a single lap from track two to fine tun the model which I considered sufficient.

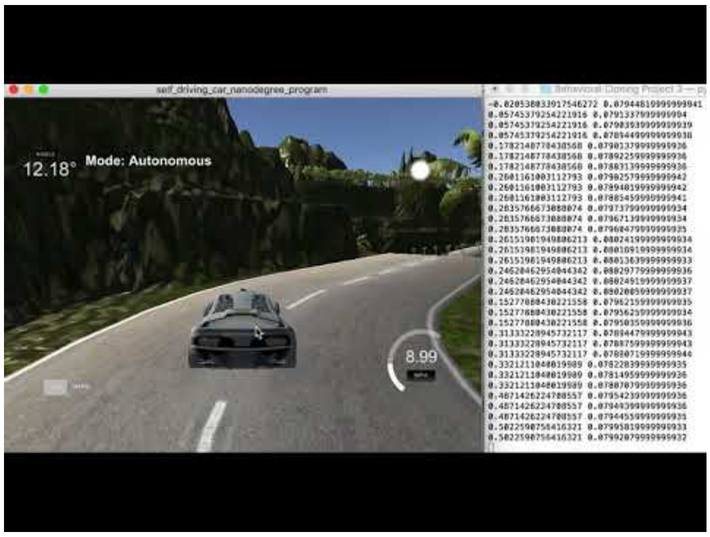
```
model = load_model(training_parameters['model_name'])
```

The I used three epochs to fine tune the model for track two. The vehicle was able to drive track two autonomously whilst staying in the centre of the road.

self_driving_car_nanodegree_program
Behavioral Cloning Project
-0.53°
Mode: Autonomous

```
-0.02426800817552665 0.07720519999999
-0.02426800817552665 0.07720239999999
-0.02426800817552665 0.07719959999999
0.0017386421095579863 0.07722739999999
0.0017386421095579863 0.07722519999999
0.0017386421095579863 0.07722299999999
-0.003573440543934703 0.07713919999999
-0.003573440543934703 0.07713539999999
-0.003573440543934703 0.07713159999999
0.00033742879331119 0.07702579999995
0.00033742879331119 0.07701999999995
0.00033742879331119 0.07701419999995
0.012847961000585123 0.07693699999995
0.012847961000585123 0.07692979999995
0.012847961000585123 0.07692259999995
0.037546181957559586 0.07685419999995
0.037546181957559586 0.07684579999995
0.037546181957559586 0.07682739999995
-0.019546810537570675 0.07676779999999
-0.019546810537570675 0.07675819999999
-0.019546810537570675 0.07674859999999
0.010000331618182056 0.07672879999996
0.010000331618182056 0.07671899999996
0.010000331618182056 0.07670919999996
-0.018307162068784237 0.07674019999999
-0.018307162068784237 0.07673119999999
-0.018307162068784237 0.07672239999999
```

```
I tensorflow/stream_executor/cuda/cuda_
negative value (-1), but there must be
I tensorflow/core/common_runtime/gpu/gp
name: GRID K520
major: 3 minor: 0 memoryClockRate (GHz)
pciBusID 0000:00:03.0
```


self_driving_car_nanodegree_program
Behavioral Cloning Project 3 — py
12.18°
Mode: Autonomous
8.99

```
-0.020538033917546272 0.07944819999999941
0.05745379254221916 0.0791337999999994
0.05745379254221916 0.0790393999999939
0.05745379254221916 0.0789449999999938
0.1782148778438568 0.0790137999999936
0.1782148778438568 0.0789225999999936
0.1782148778438568 0.0788313999999936
0.2601161003112793 0.0790257999999942
0.2601161003112793 0.0789401999999942
0.2601161003112793 0.0788545999999941
0.2835766673088074 0.0797379999999934
0.2835766673088074 0.0796713999999934
0.2835766673088074 0.0796047999999935
0.26151981949806213 0.0802419999999934
0.26151981949806213 0.0801891999999934
0.26151981949806213 0.0801363999999933
0.24620462954044342 0.0802977999999936
0.24620462954044342 0.0802491999999937
0.24620462954044342 0.0802005999999937
0.15277880430221558 0.0796215999999935
0.15277880430221558 0.0795625999999934
0.15277880430221558 0.0795035999999936
0.31333228945732117 0.0789447999999943
0.31333228945732117 0.0788759999999943
0.31333228945732117 0.0788071999999944
0.3321211048019989 0.0782283999999935
0.3321211048019989 0.0781495999999936
0.3321211048019989 0.0780707999999936
0.4871426224708557 0.0795423999999936
0.4871426224708557 0.0794939999999936
0.4871426224708557 0.0794455999999935
0.5022590756416321 0.0799581999999933
0.5022590756416321 0.0799207999999932
```